## 🔹 1. Tokens in C

**Definition**: The **smallest unit** in a C program that has meaning to the compiler.

### Types of Tokens:

1. **Keywords** – Reserved words in C (e.g., int, return)
2. **Identifiers** – User-defined names (e.g., sum, main)
3. **Constants** – Fixed values (e.g., 10, 'A', 3.14)
4. **Strings** – Enclosed in double quotes (e.g., "Hello")
5. **Operators** – +, -, \*, /, =, etc.
6. **Special Symbols** – {}, (), [], ;, #, etc.

## 🔹 1.1 Keywords

**Definition**: Reserved words used for specific tasks. Cannot be used as identifiers.

### Examples:

| Data Types | Control Flow | Others |
| --- | --- | --- |
| int, float, char, double, void | if, else, for, while, switch, break, continue | return, sizeof, struct, typedef, const |

## 🔹 1.2 Identifiers

**Definition**: The name used to identify variables, functions, arrays, structures, etc.

### Rules for Identifiers:

* Must begin with a **letter or underscore**
* Cannot be a **keyword**
* Case-sensitive (Total ≠ total)
* Can include digits and underscores (\_, 0-9, A-Z, a-z)

## 🔹 2. Data Types and Modifiers

### 🔸 Primary Data Types:

| Type | Size (Typical) | Format Specifier |
| --- | --- | --- |
| int | 4 bytes | %d |
| float | 4 bytes | %f |
| double | 8 bytes | %lf |
| char | 1 byte | %c |

### 🔸 Type Modifiers:

Used to **change size or range** of a data type.

| Modifier | Used With | Purpose |
| --- | --- | --- |
| short | int | Smaller size |
| long | int, double | Larger size |
| signed | int, char | Can be negative or positive |
| unsigned | int, char | Only positive |

unsigned int age;

long double pi;

short int count;

## 🔹 3. Input/Output (scanf and printf)

### ✅ printf() – Output

printf("Age = %d\n", age);

### ✅ 1. ****scanf("%s", str)****

* Stops reading at **first whitespace** (space, tab, newline)
* **Unsafe** for long inputs (can cause buffer overflow)
* Best for **single-word inputs**

char name[20];

scanf("%s", name); // only reads "Amit" from "Amit Kumar"

### ❌ 2. ****gets(str)**** (❗****DO NOT USE****)

* Reads until newline \n
* Allows spaces ✅
* **Unsafe**: no way to limit input size → leads to **buffer overflow**
* **Removed from C11 standard**

char name[20];

gets(name); // Reads full line, but dangerous

### ✅ 3. ****fgets(str, size, stdin)****

* Reads until newline **or** max size - 1 characters
* Stores the **newline character** (\n) at the end
* Safe, modern, and preferred method for reading strings (especially with spaces)
* Can be used after flushing newline with getchar()

char name[50];

fgets(name, 50, stdin);

name[strcspn(name, "\n")] = 0; // remove trailing newline

### 🔸 Common Format Specifiers:

| Data Type | Format |
| --- | --- |
| int | %d |
| char | %c |
| float | %f |
| double | %lf |
| string (char array) | %s |

📝 **Note**: Always use & in scanf except for strings (char name[]).

## 🏆 ****Which One Should You Use?****

| Use Case | Recommended |
| --- | --- |
| Reading strings with spaces | ✅ fgets() |
| Reading numbers | ✅ scanf() |
| Reading strings safely | ✅ fgets() |
| Writing modern, secure C code | ❌ Avoid gets(), ❌ Be cautious with scanf() |

## 🔹 4. Type Casting

### 🔸 Implicit Casting (Automatic)

* Compiler promotes smaller types to larger types automatically.

int x = 5;

float y = 2.5;

float result = x + y; // x implicitly cast to float

### 🔸 Explicit Casting (Manual)

* Done by programmer to force a conversion.

int a = 10, b = 3;

float result = (float)a / b; // Forces float division

### Type Promotion Order:

char → int → float → double

## ✅ Example Code Using All Concepts:

#include <stdio.h>

int main() {

int age; // keyword: int, identifier: age

float height, bmi;

char name[20];

printf("Enter name, age, and height in meters: ");

scanf("%s %d %f", name, &age, &height); // I/O functions

bmi = (float)age / height; // Type casting: int to float

printf("Name: %s, Age: %d, Height: %.2f, BMI (fake): %.2f\n", name, age, height, bmi);

return 0;

}